

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-250768

(43)Date of publication of application : 14.09.2000

(51)Int.Cl. G06F 9/46
G06F 9/44
G06F 13/00
G06F 15/16

(21)Application number : 2000-045445 (71)Applicant : INTERNATL BUSINESS MACH
CORP <IBM>

(22)Date of filing : 23.02.2000 (72)Inventor : AJA A APUTO

(30)Priority

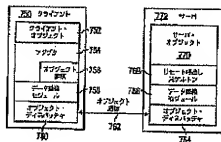
Priority number : 99 259141 Priority date : 26.02.1999 Priority country : US

(54) PROCESS METHOD AND DATA PROCESSING SYSTEM FOR CALLING METHOD OF SERVER OBJECT

(57)Abstract:

PROBLEM TO BE SOLVED: To communicate with another Java application to be executed in a non-Java environment by using an adapter generated by internal observation on an EJB interface executed on a CORBA server.

SOLUTION: Concerning a component to be used for providing a method for calling the remote method of a server object, a client 750 has a client object 752 packaged in a certain language and a server 772 has server objects 770 packaged in various languages. The client 750 is provided with an adapter 754 and the adapter 754 simulates the call of the remote method on a server object 770 by an object reference for the client object 752. The client object 752 calls a method in the adapter 754 and this method calls an object reference 756 and starts object communication with the server 772.



LEGAL STATUS

[Date of request for examination] 23.02.2000

[Date of sending the examiner's decision of rejection] 18.04.2005

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-250768

(P2000-250768A)

(43) 公開日 平成12年9月14日 (2000.9.14)

(51) Int.Cl. ⁷	識別記号	F I	特許庁 (参考)
G 0 6 F 9/46	3 6 0	G 0 6 F 9/46	3 6 0 F
9/44	5 3 0	9/44	5 3 0 M
13/00	3 5 1	13/00	3 5 1 B
15/16	6 2 0	15/16	6 2 0 T

審査請求 有 請求項の数24 O L (全 19 頁)

(21) 出願番号 特開2000-45445(P2000-45445)

(22) 出願日 平成12年2月23日 (2000.2.23)

(31) 優先権主張番号 09/259141

(32) 優先日 平成11年2月26日 (1999.2.26)

(33) 優先権主張国 米国 (US)

(71) 出願人 390009531

インターナショナル・ビジネス・マシー
ン・コーポレーションINTERNATIONAL BUSIN
ESS MACHINES CORPO
RATIONアメリカ合衆国10604、ニューヨーク州
アーモンク (番地なし)

(72) 発明者 アジャ・エイ・アプト

アメリカ合衆国78728、テキサス州オース
ティン、ナンバー2315、ウエルズ・ブラン
チ・パークウェイ 1801

(74) 代理人 100086243

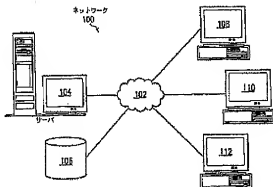
非理士 坂口 博 (外1名)

(54) 【発明の名称】 サーバ・オブジェクトのメソッドを呼び出すプロセス方法及びデータ処理システム

(57) 【要約】

【課題】 Javaクライアントが、別のJavaアプリケーションと通信することを可能にする機構を提供すること。

【解決手段】 CORBAサーバ上で実行されるEJBのインタフェース上で内蔵することによりアダプタが生成される。アダプタはJavaクライアント側に存在し、クライアントからの全てのビジネス・メソッド呼び出しをサーバ上のCORBAプロキシに委託し、JavaクライアントからCORBAプロキシへの、及びその逆のデータ整備を実行する。クライアントによりアダプタに発行されるビジネス・メソッド呼び出しは適切なデータ変換の後、アダプタによりCORBAプロキシに委託される。サーバ・オブジェクト内のメソッドを呼び出す間、クライアント・オブジェクトはアダプタ内のメソッドを呼び出し、アダプタが逐次的にオブジェクト参照を用いてサーバ上のオブジェクト参照のスケルトンを呼び出し、これが次にサーバ・オブジェクト内のメソッドを呼び出す。



【特許請求の範囲】

【請求項1】 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すプロセスであって、

前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行するステップと、

リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得するステップと、

前記オブジェクト参照をアダプタ内でラップするステップと、

前記アダプタのメソッドを呼び出すステップとを含む、プロセス。

【請求項2】 前記アダプタが前記オブジェクト参照を使用して、前記サーバ上のスケルトンのメソッドを呼び出す、請求項1記載のプロセス。

【請求項3】 前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、請求項1記載のプロセス。

【請求項4】 分散データ処理システムにおいて分散アプリケーションを実装する方法であって、

サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得するステップと、

前記プロキシをアダプタ内でラップするステップと、

前記アダプタのメソッドを呼び出すステップとを含む、方法。

【請求項5】 前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するJavaクラスである、請求項4記載の方法。

【請求項6】 前記サーバ・オブジェクトがEnterprise JavaBeanである、請求項4記載の方法。

【請求項7】 前記オブジェクト参照が命名サービスから獲得される、請求項4記載の方法。

【請求項8】 前記プロキシがCORBAプロキシである、請求項4記載の方法。

【請求項9】 前記アダプタが前記CORBAプロキシのメソッドを呼び出す、請求項8記載の方法。

【請求項10】 前記CORBAプロキシがクライアント・コンピュータ上に存在するJavaクラスである、請求項8記載の方法。

【請求項11】 前記CORBAプロキシがメソッド要求をオブジェクト・リクエスト・ブローカーに受け渡す、請求項8記載の方法。

【請求項12】 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すデータ処理システムであって、

前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する実行手段と、

リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する獲得手段と、

前記オブジェクト参照をアダプタ内でラップするラッピング手段と、

前記アダプタのメソッドを呼び出す呼び出し手段とを含む、データ処理システム。

【請求項13】 前記アダプタが前記オブジェクト参照を使用して、前記サーバ上のスケルトンのメソッドを呼び出す、請求項12記載のデータ処理システム。

【請求項14】 前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、請求項12記載のデータ処理システム。

【請求項15】 分散データ処理システムにおいて分散アプリケーションを実装するデータ処理システムであって、

サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得する獲得手段と、

前記プロキシをアダプタ内でラップするラッピング手段と、

前記アダプタのメソッドを呼び出す呼び出し手段とを含む、データ処理システム。

【請求項16】 前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するJavaクラスである、請求項15記載のデータ処理システム。

【請求項17】 前記サーバ・オブジェクトがEnterprise JavaBeanである、請求項15記載のデータ処理システム。

【請求項18】 前記オブジェクト参照が命名サービスから獲得される、請求項15記載のデータ処理システム。

【請求項19】 前記プロキシがCORBAプロキシである、請求項15記載のデータ処理システム。

【請求項20】 前記アダプタが前記CORBAプロキシのメソッドを呼び出す、請求項19記載のデータ処理システム。

【請求項21】 前記CORBAプロキシがクライアント・コンピュータ上に存在するJavaクラスである、請求項19記載のデータ処理システム。

【請求項22】 前記CORBAプロキシがメソッド要求をオブジェクト・リクエスト・ブローカーに受け渡す、請求項19記載のデータ処理システム。

【請求項23】 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すためにデータ処理システム内で使用されるコンピュータ・プログラムを記憶したコンピュータ読み取り可能な記憶媒体であって、前記コンピュータ・プログラムは、

前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する第1の命令と、

リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する第2の命令と、

リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する第2の命令と、

3

前記オブジェクト参照をアダプタ内でラップする第3の命令と、

前記アダプタのメソッドを呼び出す第4の命令とを含む、記憶媒体。

【第2項2.4】分散データ処理システムにおいて分散アプリケーションを実装するために、データ処理システム内で使用されるコンピュータ・プログラムを記述したコンピュータ読み取り可能な記憶媒体であって、前記コンピュータ・プログラムは、

サーバ・オブジェクトのプロキシのためのオブジェクト 10
参照を獲得する第1の命令と、
前記プロキシをアダプタ内でラップする第2の命令と、
前記アダプタのメソッドを呼び出す第3の命令とを含む、記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は一般に、改善された分散データ処理システムに関し、特に、分散データ処理システム内のクライアント及びサーバ上にオブジェクト 20
を含む、分散アプリケーションのための方法及び装置に
関する。

【0002】

【関連技術】本発明は、本願と同時に出願された米国特許出願"Method and System for Persisting Beans as C 30
ontainer Managed Fields" (出願人登録番号: A T 9 9 8 9 0 7) に関連する。

【0003】

【従来の技術】ソフトウェア開発者は、企業全体に及ぶアプリケーションの作成が困難であり、分散アプリケーションの作成が更に困難である根本的な問題に直面して 30
いる。更に、企業は1つのプラットフォームに閉ざされること無く、できる限り速くアプリケーションを作成することを願望する。理想的には、企業開発者は一旦アプリケーションを作成したら、それを彼らの全てのプラットフォーム上で実行することを希望する。Enterprise JavaBeans (商標) 技術は、この能力を提供しようとする。

【0004】Enterprise JavaBeans (EJB) コンポーネント・アーキテクチャは、企業がスケーラブルで安全なマルチプラットフォーム業務用アプリケーションを、 40
再利用可能なサーバ側コンポーネントとして作成することを可能にするように設計される。その目的は、企業開発者が業務論理 (またはビジネス論理) の作成だけに注力できるようにすることにより企業問題を解決することである。

【0005】サーバ側開発、及びそれをサービスするために必要とされるツールは、EJB技術のための設計目標に多大に影響する。1つの主要な設計目的は、分散アプリケーションを作成するプロセスをできる限り低減することである。この目的は、Enterprise JavaBeansの準 50

4

純な宣言属性に、通常、手作業でコーディングされる必要のあるフィーチャを削減することにより達成された。これらの宣言属性は、開発効率の多大な向上をもたらす。なぜなら、セキュリティ及びトランザクションなどの特定の振舞いがコード内にセットされるのではなく、ビーン自身上の「フラグ」であるからである。

【0006】EJB仕様は、トランザクション、セキュリティ、スレッディング、名前付け、オブジェクト・ライフサイクル、資源プーリング、リモート・アクセス、及び持続性などのシステム・レベルのプログラミングを世話するインフラストラクチャを作成する。EJB仕様はまた、既存のアプリケーションへのアクセスを単純化し、ツール作成使用のために均等なアプリケーション開発モデルを提供する。

【0007】Javaは、Javaクライアントがリモート・メソッド呼び出し (RMI) と呼ばれる方法により、別のプロセスで動作しているJavaサーバ上のメソッドを呼び出す機能を提供する。しかしながら、例えば共通オブジェクト・リクエスト・ブローカ・アーキテクチャ (CORBA: Common Object Request Broker Architecture) 準拠のサーバなど、サーバがJava環境で動作していない場合、JavaクライアントはCORBAサーバ上のメソッドに対してメソッド呼び出しを発行できない。なぜなら、JavaはCORBAオブジェクトと通信するための固有の機構を提供しないからである。

【0008】要するにCORBAは、アプリケーションがどこに配置されよう、或いは誰がそれらを設計したかに関わらず、アプリケーションが互いに通信することを可能にするオブジェクト・リクエスト・ブローカ (ORB) である。オブジェクト・リクエスト・ブローカは、オブジェクト間のクライアント-サーバ関係を確立するミドルウェアである。企業は、様々なソフトウェア・アプリケーション間の相互運用性を提供するための解決策としてCORBAに注目した。

【0009】

【発明が解決しようとする課題】データ整備、すなわち、JavaクライアントとCORBAサーバ間での異なるタイプのデータのためのデータ変換を実行するための標準的な機構は存在しない。JavaクライアントがCORBAサーバなどの非Java環境で実行されるEJBなどの、別のJavaアプリケーションと通信することを可能にする機構を有することが有利である。

【0010】

【課題を解決するための手段】本発明は、CORBAサーバ上で実行されるEJBのインタフェース上で内視 (introspect) することにより生成されるアダプタを提供する。アダプタはJavaクライアント側に存在し、EJBを実行するCORBAサーバのリモート・プロキシを含む。アダプタはEJBのビジネス・メソッドを呼

3

び出すために、EJBにより指定されるインタフェースを実装するJavaクラスである。アダプタは、クライアントからの全てのビジネス・メソッド呼び出しをサーバ上のCORBAプロキシに委託し、JavaクライアントからCORBAプロキシへの、及びその逆のデータ整備を実行する。クライアントによりアダプタに発行されるビジネス・メソッド呼び出しは適切なデータ変換の後、アダプタによりCORBAプロキシに委託される。従って、アダプタはJavaクライアントとCORBAサーバ上のEJBとの間の透過的な接合剤として作用する。

【0011】

【発明の実施の形態】 図1を参照すると、本発明が実装され得る分散データ処理システムの図を示す。分散データ処理システム100は、本発明が実現され得るコンピュータのネットワークである。分散データ処理システム100はネットワーク102を含み、これは分散データ処理システム100内で一般に接続される様々な装置及びコンピュータ間で、通信リンクを提供するために使用される媒体である。ネットワーク102は電線または光ファイバ・ケーブルなどの永久接続、または電話接続を通じて形成される一時接続を含み得る。

【0012】 図示の例では、サーバ104が配線ユニット106と共にネットワーク102に接続される。更に、クライアント108、110及び112がネットワーク102に接続される。これらのクライアント108、110及び112は、例えばパーソナル・コンピュータまたはネットワーク・コンピュータである。本発明の目的上、ネットワーク・コンピュータはネットワークに接続される任意のコンピュータであり、ネットワークに接続される別のコンピュータからプログラムまたは他のアプリケーションを受信する。図示の例では、サーバ104はポート・ファイル、オペレーティング・システム・イメージ、及びアプリケーションなどのデータをクライアント108乃至112に提供する。クライアント108、110及び112は、サーバ104のクライアントである。分散データ処理システム100は図示されない追加のサーバ、クライアント及び他の装置を含み得る。図示の例では、分散データ処理システム100はインターネットであり、ネットワーク102が、TCP/IPプロトコル方式を使用し互いに通信するネットワーク及びゲートウェイの世界的な集合を表す。主要ノードまたはホスト・コンピュータ間の高速データ通信回線の中核がインターネットの中心部にあり、データ及びメッセージを経路指定する数多くの商業用、政府用、教育用、及び他のコンピュータ・システムを含む。勿論、分散データ処理システム100は、例えばイントラネット、ローカル・エリア・ネットワーク（LAN）、または広域ネットワーク（WAN）などの、多数の異なるタイプのネットワークとしても実現され得る。図1は例と

6

して示されただけであり、本発明のプロセスの体系的な制限を意図するものではない。

【0013】 図2を参照すると、このブロック図は、図1のサーバ104など、本発明に従いサーバとして実現されるデータ処理システムを示す。データ処理システム（サーバ）200は、システム・バス206に接続される複数のプロセッサ202及び204を含む対称マルチプロセッサ（SMP）・システムであり得る。或いは、単一プロセッサ・システムが使用され得る。システム・バス206には更に、メモリ制御装置/キャッシュ208が接続され、これはローカル・メモリ209とのインタフェースを提供する。I/Oバス・ブリッジ210はシステム・バス206に接続され、I/Oバス212とのインタフェースを提供する。メモリ制御装置/キャッシュ208及びI/Oバス・ブリッジ210は、図示のように統合され得る。

【0014】 I/Oバス212に接続される周辺コンポーネント相互接続（PCI）バス・ブリッジ214は、PCIローカル・バス218とのインタフェースを提供する。多数のモデム218乃至220がPCIバス218に接続される。一般的なPCIバス・インプリメンテーションは、4つの拡張スロットまたは増設コネクタをサポートする。図1のネットワーク・コンピュータ108乃至112への通信リンクは、増設ポートを介してPCIローカル・バス218に接続されるモデム218及びネットワーク・アダプタ220を介して提供され得る。

【0015】 追加のPCIバス・ブリッジ222及び224は、PCIバス226及び228のためのインタフェースを提供し、これらのバスから追加のモデムまたはネットワーク・アダプタをサポートされ得る。このように、サーバ200は複数のネットワーク・コンピュータへの接続を可能にする。また図示のように、メモリマップド・グラフィックス・アダプタ230及びハード・ディスク232も、直接的にまたは間接的にI/Oバス212に接続され得る。

【0016】 当業者には明らかなように、図2に示されるハードウェアは変化した。例えば、光ディスク・ドライブなどの他の周辺装置も、追加で、または図示のハードウェアの代わりに使用され得る。図示の例は、本発明に関する体系的な制限を意図するものではない。

【0017】 図2に示されるデータ処理システムは、例えばIBMの製品であるIBM RISC/System6000システムであり、AIX（Advanced Interactive Executive）オペレーティング・システムを実行する。

【0018】 次に図3を参照すると、本発明が実現され得るデータ処理システムのブロック図が示される。データ処理システム300は、クライアント・コンピュータの例である。データ処理システム300は、周辺コンポーネント相互接続（PCI）ローカル・バス・アーキテ

クチャを使用する。図示の例はPCLバスを使用するが、マイクロチャネル及びISAなどの他のバス・アーキテクチャも使用され得る。プロセッサ302及び主メモリ304がPCLブリッジ308を介してPCLローカルバス306に接続される。PCLブリッジ308は、プロセッサ302のための統合型メモリ制御装置及びキャッシュ・メモリを含み得る。PCLローカルバス306への追加の接続は、直接コンポーネント相互接続を介してまたは増設ボードを介して形成され得る。図示の例では、ローカル・エリア・ネットワーク (LAN) ・アダプタ310、SCSIホスト・バス・アダプタ312、及び拡張バス・インタフェース314が直接コンポーネント接続によりPCLローカルバス306に接続される。それに対して、音声アダプタ316、グラフィックス・アダプタ318、及びスマートカード・アダプタ319は、拡張スロットに挿入される増設ボードにより、PCLローカルバス306に接続される。拡張バス・インタフェース314は、キーボード及びマウス・アダプタ320、モデム322、及び追加のメモリ324のための接続を提供する。SCSIホスト・バス・アダプタ312は、ハード・ディスク・ドライブ326、テープ・ドライブ328、及びCD-ROMドライブ330のための接続を提供する。一般的なPCLローカルバス・インプリメンテーションは、3つまたは4つのPCL拡張スロットまたは増設コネクタをサポートする。

【0019】オペレーティング・システムはプロセッサ302上で実行され、図3のデータ処理システム300内の様々なコンポーネントの制御を調整及び提供するために使用される。オペレーティング・システムはIBMから提供されるOS/2などの、市販のオペレーティング・システムであってよい。「OS/2」はIBMの商標である。Javaなどのオブジェクト指向のプログラミング・システムが、オペレーティング・システムと共に実行され、データ処理システム300上で実行されるJavaプログラムまたはアプリケーションからオペレーティング・システムに呼び出しを提供する。「Java」はサン・マイクロシステムズ社の商標である。オペレーティング・システムの命令、オブジェクト指向オペレーティング・システム、及びアプリケーションまたはプログラムは、ハード・ディスク・ドライブ326などの記憶装置上に配置され、主メモリ304にロードされ、プロセッサ302により実行される。

【0020】当業者には明らかなように、図3のハードウェアはインプリメンテーションに応じて変化する。フラッシュROM (または等価な不揮発性メモリ) または光ディスク・ドライブなどの、他の内部ハードウェアまたは周辺装置が更に追加で、または図3に示されるハードウェアの代わりに使用され得る。また、本発明のプロセッサはマルチプロセッサ・データ処理システムに適用

され得る。

【0021】例えば、データ処理システム300は、ネットワーク・コンピュータとして任意に構成される場合、図3に点線332で示されるボックス内のSCSIホスト・バス・アダプタ312、ハード・ディスク・ドライブ326、テープ・ドライブ328、及びCD-ROM330を含まなくてよい。すなわち、点線ボックス内のコンポーネントは任意に含まれることを示す。その場合、正確にはクライアント・コンピュータと呼ばれるコンピュータは、LANアダプタ310やモデム322などの、特定タイプのネットワーク通信インタフェースを含まねばならない。別の例として、データ処理システム300は自身が特定タイプのネットワーク通信インタフェースを含むか否かに関わらず、特定タイプのネットワーク通信インタフェースに頼ることなく、ブート可能に構成される独立型のシステムである。更に別の例として、データ処理システム300はパーソナル・デジタル・アシスタント (PDA) 装置であり、これはオペレーティング・システム・ファイルまたはユーザ生成データを記憶する不揮発性メモリを提供するために、ROMまたはフラッシュROMにより構成される。

【0022】図3の図示の例、及び前述の例は、本発明に関する体系的な制限を意味するものではない。

【0023】本発明は、JavaクライアントがCORBAサーバ上のEJBと通信可能な分散アプリケーションを提供する方法、データ処理システム、及び命令を提供する。分散アプリケーション内における本発明について述べる都合上、従来の総称的な分散アプリケーションについて詳述することにする。

【0024】本発明のプロセッサは、オブジェクト指向プログラミング・システムであるJavaプログラミング・システムを用いて実装され得る。オブジェクト指向プログラミング技術は、「オブジェクト」の定義、作成、使用及び命令を含み得る。これらのオブジェクトは、データ要素または属性、及びデータ要素を操作するメソッドを含むソフトウェア・エンティティである。オブジェクトはまた、オブジェクト内のメソッドをトリガまたは制御するオブジェクト外の事象に関連するデータを含み得る。

【0025】オブジェクトは、「クラス」を定義することにより定義される。クラスはオブジェクトではなくテンプレートであり、コンピュータに実際のオブジェクトの作成方法を指示する。例えば、クラスはデータ変数の数及びタイプや、データを操作する機能に関わるステップを指定し得る。オブジェクトは実際、コンストラクタと呼ばれる特殊機能によりプログラム内で作成される。コンストラクタは、対応するクラス定義、及びオブジェクト作成の間に提供される引数などの追加の情報を使用し、オブジェクトを構成する。オブジェクトはデストラクタと呼ばれる特殊機能により破壊される。Javaは

また、インタフェースとして知られる完全な抽象クラスの作成を許容し、これは他のクラスが如何にメソッドを処理しているかに関わり無く、複数のクラスと共用され得るメソッドの定義を可能にする。

【0026】図4を参照すると、従来の分散アプリケーションを示す図が示される。図示のように、クライアント・オブジェクト400は、分散アプリケーションのクライアント側に存在し、一方、サーバ・オブジェクト402は分散アプリケーションのサーバ側を形成する。クライアント・オブジェクト400は、例えば図1の分散データ処理システム100内のクライアント108などの、クライアント・コンピュータ上に配置される。サーバ・オブジェクト402は、図1のサーバ104などのサーバ上に配置される。クライアント・オブジェクト400は、サーバ・オブジェクト402内に実装される様々なビジネス規則またはビジネス論理にもとづき、データベース404をアクセスするために、サーバ・オブジェクト402への呼び出しを開始する。データベース404はサーバ内に配置されるか、リモート・データベースである。サーバ・オブジェクト402は現企業アプリケーション410へのアクセス、及びレガシ・アプリケーション408へのアクセスも提供し得る。分散アプリケーションを作成するユーザが、サーバ・オブジェクト402を実装する第2階級(tier)のコンピュータの、ホスト名またはインターネット・プロトコル(IP)・アドレスを構成することを可能にするためにカスタマイザが提供され得る。この例では、Javaリモート・メソッド呼び出し(RMI)プロトコル408が、クライアント・オブジェクト400及びサーバ・オブジェクト402などのオブジェクト間の分散通信のために使用される。

【0027】サーバ・オブジェクト402は、アプリケーション・プログラミング・インタフェース(API)を用いて実装される実際のビジネス論理を含む。APIはJava定義のJavaデータベース接続性(JDBC)構造化照会言語(SQL)データベース・アクセス・インタフェースを使用し、これは広範囲のレルショナル・データベースへの均等なアクセスを提供する。図示の例では、これらのデータベースはデータベース404内で見いだされ得る。サーバ・オブジェクト402は、クライアント・オブジェクト400から呼び出される必要な機能を提供するために使用されるメソッドを含む。前述のメソッドは、他の後置システム(すなわちCICS、IMS、MQ、SAPなど)をアクセスするために作成され、JDBCまたはデータベース・アクセスだけに制限されるべきではない。

【0028】分散アプリケーション内におけるデータベース・アクセスの例について、引き続き述べると、データベース・アクセス機能は2つのオブジェクト、すなわちクライアント・オブジェクト及びサーバ・オブジェク

トに分けられる。サーバ・オブジェクトは、クライアント・オブジェクトから呼び出され、JDBCを使用するメソッドを含むインタフェースを実装する。

【0029】クライアント・オブジェクトは、様々な事象の事象ソースまたは事象シンクとして作用する。生成時、クライアント・オブジェクトはクライアント側ビルダ環境で使用され、他のクライアント側ソフトウェア・コンポーネントに接続される。例えば、クライアント側生成時にアプリケーション・アセンブラがクライアント側ビルダ環境を使用し、GUIオブジェクトをクライアント・オブジェクトに接続する。従って、ボタンが押下されるとき、特定のデータを検索するための事象がオブジェクトに送信される。サーバ側生成時には、アプリケーション・アセンブラがビルダ環境を使用し、サーバ・オブジェクトを接続する。

【0030】実行時、ユーザはクライアント上のボタンを押下し、これは特定のデータが表示のために要求されることを示す。GUIオブジェクトは事象を生成し、これがクライアント・オブジェクトに送信される。クライアント・オブジェクトは、必要なデータを要求するサーバ・オブジェクト上のメソッドを呼び出す。クライアント・オブジェクトは、リモート・メソッド呼び出し(RMI)またはIICP(Internet Inter-Obj Protocol)を使用するオブジェクト・リクエスト・ブローカ(ORB)などのプロトコルを使用することにより、その対応するサーバ・オブジェクトと通信する。サーバ側オブジェクトはデータを検索し、データをクライアント側オブジェクトに返送する。データが次にクライアント側オブジェクトによりGUIオブジェクトに返送され、エンドユーザに表示される。

【0031】図示の例では、クライアント・オブジェクト400及びサーバ・オブジェクト402を実装するJavaビーンズが使用され得る。ビーンズは純粋なオブジェクトと区別するのは、それが両性インタフェースと呼ばれる外部インタフェースを有することであり、これはツールがコンポーネントが実行しようとしていることを検出し、それを他のビーンにフック留めし(hook up)、それを他のビーンにプラグ・インすることを可能にする。2つの異なるタイプのビーン、すなわちJavaBean及びEnterprise JavaBeans(EJB)が使用され得る。JavaBeansは単一のプロセスに対して局所的であるようにもくろまれ、しばしば実行時に見ることができ、このビジュアル・コンポーネントは、ボタン、リスト・ボックス、グラフィックまたはチャートなどである。

【0032】EJBは、サーバ上で実行され、クライアントにより呼び出されるように設計される、見ることができないリモート・オブジェクトである。EJBは複数の見ることができないJavaBeansから構成される。EJBはあるマシン上に集約し、別のマシンから遠隔的に呼

11

ひ出されるようにもくろまれる。EJBは、ツールにより脱出され得るビーンに関する記述としてもくろまれる展開記述子 (deployment descriptor) を有する。EJBはまたプラットフォーム独立であり、Javaをサポ

ートする任意のプラットフォーム上で使用され得る。
 【0033】サーバ・ビーンズすなわちEJBは、サーバ上に展開される適用的に実行可能なコンポーネントまたはビジネス・オブジェクトである。EJBはそれらが適用的にアクセスされることを可能にするプロトコルを有し、このプロトコルは、EJBが特定のサーバに導入または展開されることを可能にする。EJBはサービスの主要な品質、セキュリティ、トランザクション振舞い、同時性 (2つ以上のクライアントにより一度にアクセスされる能力)、及び持続性 (どのようにそれらの状態が保管され得るか) をEJBサーバ上においてそれらが配置されるコンテナに委託する一連の機構を有する。EJBはそれらの振舞い、異なるサービス品質を提供するコンテナ内に納入される。展開ツールの使用を通じて、プラットフォーム独立のJavaBeanがプラットフォーム特定のEJBに取り入れられる。後者は、既存のビ

ネス・システム及びアプリケーションの特定の要求に合致するために使用可能な適正なサービス品質を有する。
 【0034】クライアント・ビーン400とサーバ・ビーン402間のこの分離により、サーバ・ビーン402内の様々なビジネス論理の変更がクライアント・ビーン400の変更無しに実行され得る。これは単一のサーバをアクセスする数多くのクライアントが存在し得ることを考慮すると望ましい。更にこれらのプロセスは、例えばCORBAなどの今日のでないプログラミング言語により作成されたプログラムにも適用され得る。こうしたプログラムの動的変更は、そのプログラムをJavaなどのオブジェクト指向プログラミング・システムと互換にするインタフェースを作成することにより実行され得る。

【0035】分散アプリケーション内及び間の通信サービスは、例えばOMG (Object Management Group) コンソーシアムにより設計されたCORBA規格など、Javaリモート・メソッド呼び出し (RMI) 以外の他のタイプの分散プロトコルによっても同様に実装され得る。CORBAは、今日使用可能な急増しつつあるハードウェア及びソフトウェア製品の間の相互運用性のニーズに対するOMGの回答である。要するに、CORBAはアプリケーションがどこに配置されようと、また誰がそれらを設計したかに関わらず、アプリケーションが互いに通信することを可能にするオブジェクト・リクエスト・ブローカ (ORB) である。

【0036】オブジェクト・リクエスト・ブローカは、オブジェクト間のクライアント-サーバ関係を確立するミドルウェアである。オブジェクト・リクエスト・ブローカを使用することにより、クライアントは同一マシン

12

上に存在する、またはネットワークを介して存在するサーバ・オブジェクト上のメソッドを遠隔的に呼び出すことができる。オブジェクト・リクエスト・ブローカは呼び出しを格納し、要求を処理するオブジェクトを思い出す責任があり、それにパラメータを受け渡し、そのメソッドを呼び出し、結果を返却する。クライアントは、オブジェクトが配置されている場所、そのプログラミング言語、そのオペレーティング・システム、またはオブジェクトのインタフェースの一部でない他のシステム属性を知る必要はない。そうすることにより、オブジェクト・リクエスト・ブローカは、異種の分枝環境内の異なるマシン上のアプリケーション間の相互運用性を提供し、複数のオブジェクト・システムを縦横目無く相互接続する。

【0037】典型的なクライアント/サーバ・アプリケーションを扱うために、開発者は彼ら自身の設計または認識された規格を使用し、装置間で使用されるプロトコルを定義する。プロトコル定義はインプリメンテーション言語、ネットワーク・トランスポート、及び接続の他のファクタに依存する。オブジェクト・リクエスト・ブローカはこのプロセスを単純化し、柔軟性を提供する。オブジェクト・リクエスト・ブローカは、構成中のシステムの各コンポーネントのために、プログラマが大幅の適切なオペレーティング・システム、実行環境、及びプログラミング言語さえも選択し、使用することを可能にする。更に重要な点は、オブジェクト・リクエスト・ブローカは既存のコンポーネントの統合を可能にする。ORBベースの解決策では、開発者は新たなオブジェクトを作成するために使用するのと同一のインタフェースを用いて、レガシ・コンポーネントをモデル化し、次に標準化バスとレガシ・インタフェース間を要変する「ラッパー (wrapper)」* コードを作成する。

【0038】CORBAは、オブジェクト指向標準化及び相互運用性に向けての重要な進歩を表すオブジェクト・リクエスト・ブローカである。CORBAにより、ユーザは、情報が必要のソフトウェアまたはハードウェア・プラットフォーム上に存在するか、或いは情報がネットワーク内のどこに配置されるかを知る必要無しに、情報へのアクセスを遠隔的に獲得できる。CORBAオブジェクト開発の目的は、オブジェクト・サーバまたは単にサーバの生成及び登録である。サーバはプログラムであり、1つ以上のオブジェクト・タイプのインプリメンテーションを含み、オブジェクト・リクエスト・ブローカに登録される。

【0039】CORBAは、オブジェクトが企業及び大階さえも横断して通信することを可能にするオブジェクト・バスを詳述する。CORBAは、高機能コンポーネントが互いに発見し合い、オブジェクト・バスを介して相互運用することを可能にするように設計された。しかしながら、CORBAは単なる相互運用性を超越する。

50

CORBAはバス関連サービスの拡張セットを指定することにより、オブジェクトを作成及び消去し、それらを名前によりアクセスし、それらを永久的記憶装置に記憶し、それらの状態を外部化し、それらの間の臨時の関係を設定する。

【0040】JDKバージョン1.1の発表により、Javaはリモート・メソッド呼び出し(RMI)と呼ばれる、それ自身の組み込み型の固有のオブジェクト・リクエスト・ブローカを有する。リモート・メソッド呼び出しは、リモート・オブジェクト上のメソッド呼び出しを行うといった総称的な意味では、オブジェクト・リクエスト・ブローカであるが、それはCORBA準拠のオブジェクト・リクエスト・ブローカではない。リモート・メソッド呼び出しはJavaに固有である。リモート・メソッド呼び出しは、本来、中核を成すJava言語の拡張である。リモート・メソッド呼び出し(RMI)は、Javaオブジェクト直列化、移植性のあるダウンロード可能なオブジェクト・インプリメンテーション、及びJavaインタフェース定義などの、他の多くのライブラリに依存する。他方、リモート・メソッド呼び出しは幾つかの困難を有し、その最も偉大な強み、すなわちJavaとの堅い統合の結果である原理上の制限が、他の言語により作成されたオブジェクトまたはアプリケーションとの使用を非現実的なものにする。

【0041】Javaは、拡張によるリモート・メソッド呼び出しと共に具体的なプログラミング技術である。Javaは基本的に、実行可能コードを作成及び編成する問題を解決するために設計された。Javaはそれなりにプログラミング技術の間に、特定のポイントを突き上げる。Javaと他のプログラミング言語との間に存在する隔たりは、時に行き交うことが困難である。例えば、JavaコードからAdaコードに呼び出しを行うために使用される技術は、JavaコードからC++コードに呼び出しを行うために使用されるものと幾分異なる。このことは多言語環境において、システムの生成を複雑化させ、複雑化は使用される言語の数と共に著しく、時に非線形的に増加する。

【0042】JavaはJavaネイティブ・インタフェース(JNI:Java Native Interface)と呼ばれるAPIを提供し、これはJavaコードが他の言語によるルーチンを実行出す、または呼び出されることを可能にする。JNIは主にC及びC++言語との相互運用のために適合化され、習得するのにかなり難しいインタフェースである。リモート・メソッド呼び出しはJava間の技術である。Javaクライアントがリモート・メソッド呼び出しを使用し、別の言語のリモート・オブジェクトと通信したい場合、「異国の(foreign)」リモート・オブジェクトと一緒に配置されるJava媒介を介する必要がある。この場合の根本的な問題は、Javaが定義上、言語自身の境界内で作用するプログラミング

技術であることである。

【0043】それに対してCORBAは統合技術であり、プログラミング技術ではない。CORBAは特に、異種のプログラミング技術を一括に結合する接合剤として設計される。CORBAはプログラミング空間内のポイントとして存在するのではなく、個々の言語を表すポイント間の空間を占有する。例えば、JavaクライアントがCORBA技術を使用し、C++オブジェクトと通信するとき、C++プログラマ及びJavaプログラマの両者は、完全にそれぞれの言語環境内で作業する。CORBAのオブジェクト・リクエスト・ブローカは、JavaクライアントにJavaスタブ・インタフェースを提供し、C++プログラマにC++スケルトン・インタフェースを提供する。CORBAは言語間問題を自動的に解決する。

【0044】CORBAは統合指向の視地を提供し、そこでは設計努力がシステムの要請の境界に絞られる。基礎となるインタフェース技術(例えばI/O)は、それらの境界をできる限り柔軟で、適応的で、プログラミング技術に独立するように設計される。CORBAなどのインタフェース技術は、プログラミング技術よりも長い半減期を有するだけでなく、廃れたプログラミング言語への依存による、アプリケーションの追加及び死去に対する最善の防御である。

【0045】図5を参照すると、CORBA規格を使用する従来の分散アプリケーションが示される。クライアント・オブジェクト500は、通信リンク510として示されるI/O規格を使用し、CORBAサーバ520と通信する。CORBAサーバ520は、異種のタイプのソフトウェア・オブジェクトのための統合及び相互運用性を提供する。JavaBean521、C++オブジェクト522、EnterpriseJavaBean(EJB)523、及びCOBOLオブジェクト524は、サーバ520により提供されるCORBAサービスを介して通信し、サービス機能を登録する。クライアント・オブジェクト500は、CORBAサービスを通過して、オブジェクト521乃至524により提供される機能及びメソッドを呼び出す。

【0046】図6を参照すると、CORBAを用いて相互運用性機能を提供する従来の分散アプリケーション内のコンポーネントが示される。図6は、クライアント602から、サーバ内のCORBAオブジェクト・インプリメンテーションに送信されるメソッド要求618を示す。クライアントは、CORBAサーバ上のメソッドを呼び出す任意のコードであり、ことによるとそれ自身がCORBAオブジェクトである。サーバ620は、オブジェクト・インプリメンテーションのインスタンス、すなわちCORBAオブジェクトを実現する実際のコード及びデータである。

【0047】CORBAサーバ・オブジェクトのクライ

15

アント602は、サーバ・オブジェクトに対するオブジェクト参照616を得し、クライアントはこのオブジェクト参照を使用し、メソッド要求618を発行する。

【0048】オブジェクト参照は、オブジェクト・リクエスト・ブローカ内のオブジェクトを指定するために必要とされる情報である。クライアントは通常、幾つかの異なる方法によりオブジェクト参照を獲得する。第1に、クライアントはオブジェクトを作成するためにオブジェクト上の作成メソッドを呼び出す。作成メソッドは、新たなオブジェクトに対するオブジェクト参照をク

ライアントに返却する。第2に、クライアントは、命名サービスに要求を発行することにより、オブジェクト参照を獲得する。命名サービスはオブジェクト参照を名前

によりデータ構造内に記憶し、クライアントは特定タイプのハードコード化ネットワーク・アドレスではなし

に、オブジェクトの問題付けられる名前によりオブジェクト参照を探索または解析する。すなわち、オブジェクトを同一の物理マシン内で、またはネットワーク上のどこかで突き止める。最後に、クライアントは、オブジェクト参照をストリング化することにより特定の作成さ

れたストリングからオブジェクト参照を獲得する。

【0049】一旦オブジェクト参照が獲得されると、クライアントはCORBAオブジェクトを自身上に呼び出せるように、適切なタイプに制限する。

【0050】サーバ・オブジェクトが遠隔的である場合、オブジェクト参照はスタブ機能604を指し示し、これがオブジェクト・リクエスト・ブローカ・マシンを

使用して、呼び出しをサーバ・オブジェクトに転送する。CORBAクライアントは、その全てのデータ整備

及びI/O作業を実行するために、ローカル・オブジェクト・リクエスト・ブローカ・オブジェクトを必要と

する。スタブ・コードはオブジェクト・リクエスト・ブローカ606にあり、サーバ・オブジェクトを実行する

マシンを識別し、そのマシンのオブジェクト・リクエスト・ブローカ610に、オブジェクトのサーバ614へ

の接続を依頼する。スタブ・コードが接続を有するとき、それはオブジェクト参照及びパラメータを宛先オブ

ジェクトのインプリメンテーションにリンクされるスケルトン・コード612に送信する。スケルトン・コードは呼び出し及びパラメータを要求されたインプリメン

テーション特定の形式に変換し、オブジェクトを呼び出す。あらゆる結果または例外が、同一の経路に沿って返

却される。

【0051】クライアントは、CORBAオブジェクトの位置、インプリメンテーション詳細、及びオブジェクトをアクセスするために使用されるオブジェクト・リク

エスト・ブローカを知らない。異なるオブジェクト・リクエスト・ブローカが、I/O608を介して通信し

得る。

【0052】クライアントは、CORBAオブジェクト

16

のインタフェース内で指定されるメソッドだけを呼び出し得る。インタフェースはオブジェクト・タイプを定義し、命名メソッド及びパラメータのセットの他に、これらのメソッドが返却し得る例外タイプを指定する。サーバ・オブジェクトのクライアントは、サーバ・オブジェクトに対応するオブジェクト参照へのアクセスを有し、そのオブジェクトに対するオペレーションを呼び出す。

クライアント・オブジェクトは、そのインタフェースに従い、サーバ・オブジェクトの論理構造だけを知り、メソッド呼び出しを通じてサーバ・オブジェクトの操縦に

に精通する。重要な点は、クライアント・サーバ関係が2つの特定のオブジェクトに関連することである。すなわち、一方のサーバ・オブジェクトのインプリメンテーションは、他のサーバ・オブジェクトのクライアントで

有る得る。

【0053】スタブ及びスケルトン・ファイルは、様々な方法により生成される。スタブ・ファイルは、クライアント・プログラミング言語により、クライアントにサーバ・メソッドへのアクセスを提供する。サーバ、スケルトン・ファイルは、オブジェクト・インプリメンテーションをオブジェクト・リクエスト・ブローカ（ORB）実行時に結び付ける。オブジェクト・リクエスト・ブローカはスケルトンを使用し、メソッドをオブジェクト・インプリメンテーション・インスタンス（サーバ）にディスパッチする。

【0054】従来技術の説明から本発明の説明に目を向けると、図7及び図8は、Java及びCORBAの利点を結合する本発明の詳細を示す。前述のように、リモート・メソッド呼び出しは、Javaクライアントがリモート・メソッド呼び出しによりリモート・オブジェクトと通信することを強要する、Java間の技術である。Javaは定数上、言語自身の境界内で作用するプログラミング技術である。それに対して、CORBAは統合技術であり、プログラミング技術ではない。本発明は、JavaクライアントがCORBAクライアント上に存在するEJBのビジネス・メソッドを呼び出すことを可能にする。

【0055】図7を参照すると、このブロック図は、CORBAサーバ内で実行されるEnterprise JavaBean（EJB）のリモート・ビジネス・メソッドを呼び出す方法を説明するために使用されるコンポーネントを示す。図7のシステムは、図4及び図6に示されるシステムに類似する。図7は、クライアント700及びサーバ726が、Java仮想マシンJVM702及びJVM724を含むJava実行時環境によりインテールされる以外は、図4に示される分散アプリケーションと類似する。図7は図6と類似する。なぜなら、クライアント602及びサーバ614と同様に、Javaクライアント・オブジェクト704がリモートEJB728内のメソッドを呼び出すおとしていからである。

50

【0056】更に、図7の分散アプリケーションは、オブジェクト・リクエスト・ブローカORB714及びORB718を含み、オブジェクト要求及び応答をIOP716を介して伝送する。図6の総体的なアーキテクチャは、クライアント、サーバ、スタブ、及びスケルトンを含み、オブジェクト・リクエスト・ブローカが図7のソフトウェア・アーキテクチャ内に反映され、これは本発明の方法に従い、JavaオブジェクトがCORBAを用いて通信することを可能にする。

【0057】JVM702は、EJB728内に存在するビジネス・メソッドを呼び出そうとしているクライアント・オブジェクト704を含む。Java仮想マシン(JVM)は、メモリ内に存在する仮想コンピュータ・コンポーネントである。特定のケースでは、JVMはプロセッサ内で実行される。JVMはJavaプログラムが、コードがコンパイルされたあるプラットフォームだけではなく、異なるプラットフォーム上で実行されることを可能にする。JavaプログラムはJVMのためにコンパイルされる。このように、Javaは、様々な中央処理ユニット及びオペレーティング・システム・アーキテクチャを含み得る多くのタイプのデータ処理システムのためにアプリケーションをサポートできる。

【0058】Javaアプリケーションが異なるタイプのデータ処理システム上で実行されるようにするために、コンパイラは一般に、アーキテクチャ中立ファイル形式を生成する。すなわち、Java実行時システムが存在する場合、コンパイルされたコードが多くのプロセッサ上で実行可能である。Javaコンパイラは、特定のコンピュータ・アーキテクチャには特定のバイナリコード命令を生成する。バイトコードは、Javaコンパイラにより生成されるマシン独立なコードであり、Javaインタプリタにより実行される。Javaインタプリタは、バイトコードを交互に解釈及び実行するJVM内のモジュールである。これらのバイトコード命令は、任意のコンピュータ上で容易に解釈され、また、固有マシン・コードに即座に容易に変換されるように設計される。

【0059】JVM702は、EJBインタフェース708をクライアント・オブジェクト704に提供するアダプタ706を含み、それにより、クライアント・オブジェクト704はリモート・メソッド呼び出しの既知のメソッドを使用し、EJBインタフェース708内のメソッドを呼び出す。EJBインタフェース708はCORBAプロキシ710を呼び出し、サーバ726とのCORBA通信を開始する。CORBAプロキシ710は、オブジェクト要求をオブジェクト・リクエスト・ブローカORB714に受け渡す。この例では、ORB714はJavaにより実現される。ORBがC++により実現される場合、オブジェクト要求はJavaネイティブ

イブ・インタフェース(JNI)を通じて受け渡される。ORB714及びORB718はIOP716を介して通信し、オブジェクト要求がクライアント及びサーバ・オブジェクトにより、オブジェクト・リクエスト・ブローカを用いて自動的にサポートされることを保証する。一旦ORB718がオブジェクト要求を受信すると、EJBスケルトン720内のコードが呼び出され、EJB728から要求されたビジネス・メソッドの呼び出しを開始する。EJBスケルトン720はJNI722を用いてEJB728を呼び出し、適切な引き数をJVM724内に含まれるEJB728に受け渡す。この特定の例では、EJBスケルトン720はC++により実現されるが、JNI722の使用を要求するJava以外の別の言語によっても実現される。

【0060】図8を参照すると、このブロック図は、サーバ・オブジェクトのリモート・メソッドを呼び出す方法を実現するために使用されるコンポーネントを示す。図8のシステムは、クライアント750がある言語により実装されるクライアント・オブジェクト752を有し、サーバ772が異なる言語により実装されるサーバ・オブジェクト770を有するように、環境が一般化される以外は、図7に示されるシステムと類似する。

【0061】クライアント750はアダプタ754を含む。アダプタはクライアント・オブジェクト752のためにオブジェクト参照により、サーバ・オブジェクト770のリモート・メソッド呼び出しをシミュレートする。クライアント・オブジェクト752はアダプタ754内のメソッドを呼び出し、これがオブジェクト参照756を呼び出し、サーバ772とのオブジェクト通信を開始する。オブジェクト参照756は、データ転送モジュール758を通じてオブジェクト要求をオブジェクト・ディスパッチャ760に受け渡す。オブジェクト・ディスパッチャ760及びオブジェクト・ディスパッチャ764は、オブジェクト通信リンク762を介して通信し、オブジェクト要求がクライアント及びサーバ・オブジェクトにより適切なオブジェクト要求プロトコルを用いて、自動的にサポートされることを保証する。オブジェクト・ディスパッチャ764はオブジェクト要求をデータ転送モジュール766を介して、リモート呼び出しスケルトン768に受け渡す。データ転送モジュール758及びデータ転送モジュール768は、異なる言語により実装されるオブジェクトを含む相互運用環境のために、適切なデータ変換及び呼び出し変換機構を提供する。リモート呼び出しスケルトン768は、サーバ・オブジェクト770内のメソッドを呼び出し、適切な引き数をサーバ・オブジェクト770に受け渡す。

【0062】図9を参照すると、CORBAサーバ内で実行されるEnterprise JavaBean(EJB)上のリモート・ビジネス・メソッドを呼び出す方法を示すフロー図が示される。図7に関連して前述したように、Java

19

クライアント・オブジェクトからJavaアダプタ、ORB及びEJBスケルトンを介して所望のEJBに至るプロセス・フローは、図9に示されるプロセス・フローに類似する。図9は、本発明のJava及びCORBA環境内の実際の呼び出し及びクラス操作を示す。

【0063】プロセスは、クライアント内のコードがリモート・オブジェクトの名前を探索するとき開始する(ステップ802)。クライアントは、EJB内のビジネス・メソッドなど、リモート・サーバ内のオブジェクトのメソッドの実行を要求する。ルックアップ・メソッド内のコードが所望のメソッドを含むリモート・オブジェクトに対応するCORBAプロキシの名前を見いだす(ステップ804)。適切なCORBAプロキシが見つめられた後、その対応するCORBAプロキシのための適切なアダプタ・クラスが突き止められる(ステップ806)。CORBAプロキシは適切なアダプタ・クラスによりラップ(wrap)され(ステップ808)、アダプタ・クラスが呼び出し側クライアント・コードに総称オブジェクトとして返却される(ステップ810)。クライアント・コードは、返却された総称オブジェクトを必要なクラス・タイプに固定する(ステップ812)。クライアント・コードは次に、新たに取得されたオブジェクトから所望のメソッドを呼び出し(ステップ814)。新たに取得されたアダプタ・クラス内の所望のメソッドが呼び出される(ステップ816)。呼び出されたアダプタ・クラス・メソッド内のコードは、CORBAプロキシ内のその対応するメソッドを呼び出す(ステップ818)。一旦メソッドがCORBAプロキシ内で実行を開始すると、CORBAインフラストラクチャが、CORBAサーバ上のCORBAプロキシに対応するリモート・オブジェクトの呼び出しを処理する(ステップ820)。

【0064】図9の方法は、図8に示されるオブジェクト指向環境のために一般化され得る。CORBAプロキシを通じてEJBのメソッドを呼び出すのではなく、ソース・オブジェクト及びターゲット・オブジェクトがサーバ・オブジェクトのメソッドに一般化され、そのオブジェクト参照を通じて呼び出される。本発明の一般化された方法によれば、サーバ・オブジェクトに対するオブジェクト参照がアダプタによりラップされ、クライアント・オブジェクトによるサーバ・オブジェクト内のメソッドの呼び出しがアダプタにより透過的に処理される。クライアント・オブジェクトがサーバ・オブジェクトのメソッドを呼び出そうとするとき、メソッドは実際、アダプタ・オブジェクト内で呼び出される。アダプタ・クラスは本質的に、オブジェクト参照がクライアント・コードから分離されるように、オブジェクト参照をラップする。クライアントはオブジェクト参照について何も知らない。クライアントはただ、アダプタ・コードに「話しかける(talk to)」だけであり、他方、サーバのスケルトン・コードに「話しかける」オブジェクト参照は、クライアント・ベースのアダプタに関して何も知らない。

20

【0065】図10乃至図13を参照すると、JavaクライアントがCORBAサーバ内で実行されるInterprise JavaBeanから、リモート・ビジネス・メソッドを呼び出す分散アプリケーションを記述するJavaプログラミング言語ステートメントの例が示される。図10では、Javaプログラム内の標準のリモート・メソッド呼び出し(RMI)技術の例が示される。CustomerImplクラスのリモート・オブジェクトは、インタフェース「CustomerInterface」により定義されるビジネス・メソッドのセットを有し得る。CustomerInterface内に含まれるメソッドは、Javaクライアントから遠隔的に呼び出され得る。この場合、クライアント・コードはステートメント902乃至906に類似するとと思われる。ステートメント902は、リモート・オブジェクトのオブジェクト参照を取得するための命名サービスの使用を示す。オブジェクトobjは、CustomerInterfaceを実装するリモートJavaオブジェクトのRMIプロキシである。ステートメント902でオブジェクト参照を取得後、ステートメント904で、オブジェクトを適切なオブジェクト・タイプに固定することによりオブジェクトが制限される。ステートメント906では、Javaクライアント・コードがプロキシ・オブジェクト上のビジネス・メソッドをあたかもローカル・オブジェクトのように呼び出す。クライアントは、呼び出しがオブジェクト・リクエスト・ブローカ(ORB)を用いて実施されることを知らない。プロキシ・オブジェクトはCORBA規格により定義されるように、メソッド呼び出しをリモート・オブジェクトに転送する。

【0066】本発明の方法によれば、リモート・オブジェクトを実行するために、Java内のRMI環境が、CORBA機構を用いてシミュレートされる。特殊なNamingContext及びアダプタ・クラスのセットが、図11乃至図13に示されるように実装される。特殊なNamingContextクラス内のルックアップ・メソッドが、図11に示されるように実装される。ステートメント912は、特殊なルックアップ・メソッドの定義を示す。ステートメント914は、CORBA命名サービスのルックアップ・メソッドがステートメント914内の所望のリモート・オブジェクトに対応するCORBAプロキシを見いだすために使用されることを示す。一旦CORBAプロキシが見つめられたと、ステートメント916が新たなアダプタ・クラスを作成する。ステートメント918は、CORBAプロキシが適切なアダプタ・クラスによりラップされることを示す。ステートメント920は、ルックアップ・メソッドからの戻り値として、オブジェクト参照の返却を示す。

【0067】図12は、図10に示されるRMIに類似

50

のシミュレート化RMIのためのクライアント・コードの例を示す。ステートメント922は、リモート・オブジェクトの名前によるリモート・オブジェクトのルックアップすなわち探索を示し、図11に示されるコードにより実装された新たなNamingContextクラスを使用する。一旦オブジェクト参照が返却されると、ステートメント924がオブジェクト参照がCustomerInterfaceクラスを用いて制限されるか、固定されなければならないことを示す。ステートメント926は、クライアント・コード内で呼び出される所望のビジネス・メソッドを示す。しかしながら、CustomerInterfaceクラスの"cust"インスタンス上で呼び出されるビジネス・メソッドは、実際には、新たに定義されたNamingContextクラスにより返却されたアダプタ・オブジェクト上で呼び出される。

【0068】図13は、ビジネス・メソッドのためのアダプタ・クラス内のコードを示す。ステートメント930はbusinessMethodメソッドの定義の開始を示す。ステートメント932は、アダプタ・コードが所望のメソッド呼び出しをCORBAプロキシに委譲することを示す。この場合、プロキシはCORBAサーバ上のEJBのスケルトンのための、CORBAプロキシのJavaクラス・インプリメンテーションである。

【0069】アダプタ・クラスは、CORBAサーバ上のEJBによりサポートされるビジネス・インタフェースを実装しなければならない。アダプタ・クラスは本来、CORBAプロキシがJavaクライアント・コードから分離されるように、CORBAプロキシをラップする。重要な点は、JavaクライアントがCORBAプロキシに関して何も知らないことである。JavaクライアントはJavaベースのアダプタ・コードにだけ話しかけ、他方、CORBAサーバ上のスケルトン・コードに話しかけるCORBAプロキシは、Javaベースのアダプタに関して何も知らない。

【0070】図11乃至図13に示される例では、リモート・オブジェクトのビジネス・メソッドのための引き数、単にアダプタからプロキシ・オブジェクトに受け渡される。引き数に対するデータ変換は示されない。ビジネス・メソッドがEJBを引き数として受け取ったり、返却する場合、必要に応じて好適なアダプタにより、プロキシ・オブジェクトの適切なラップまたはラップ解除を実行することが、そのビジネス・メソッドのアダプタ・コードの義務である。図示の例では、戻り値を期待すること無く、ビジネス・メソッドがCustomerInterfaceクラスのために呼び出される。

【0071】ビジネス・メソッドが引き数として整数及びEJBを受け取り、戻り値としてEJBを返却する例では、EJBはラップ及びラップ解除されなければならない。

【0072】図14乃至図16を参照すると、Javaプログラミング言語ステートメントの例が、Javaメ

ソッドへの引き数として使用されるEJBをラップ及びラップ解除するプロセスを示す。図14は、Javaビジネス・メソッドの宣言の例であり、これは整数及びタイプ"Employee"のEJBを引き数として受け取り、タイプ"Customer"のEJBを戻り値として返却する。

【0073】図15は、リモートEJB内のビジネス・メソッドに対するシミュレート化RMIの例を示す。図11乃至図13に示されるリモートEJBの場合同様、図15は前述のように、ビジネス・メソッドの引き数をラップ及びラップ解除する追加のステップを示す以外は、プロキシをアダプタによりラップ及びラップ解除する本発明の方法を使用する。ステートメント1010は"EmployeeName"の探索、及びオブジェクト"obj1"へのその割当てを示し、ステートメント1012は、オブジェクト"obj1"から、タイプ"Employee"のオブジェクト"ee"への制限を示す。ステートメント1014は"CustomerName"の探索を示し、ステートメント1016は、Customerクラス内のビジネス・メソッドの呼び出しを示し、EJB引き数"ee"及び新たな顧客"newCustomer"の戻り値を有する。

【0074】EmployeeのEJBを引き数として受け渡すことにより、ビジネス・メソッドがJavaクライアントによりアダプタ上で呼び出されると、"Employee"インタフェースを実装するのはアダプタ・オブジェクトであり、アダプタ・コードがアダプタを"Employee"からラップ解除し、その内部のCORBAプロキシをORBを介して受け渡す。同時に、ORBを介して返却される結果の戻り値が実際、CustomerのEJBのCORBAプロキシを含む。この場合、アダプタ・コードは返却されたプロキシをクライアントに返送する前に、それを適切なアダプタによりラップする。

【0075】図16は、CustomerAdapterクラス内のアダプタ・コードの例を示し、これはEJB引き数を必要に応じてラップ及びラップ解除するためのステップを示す以外は、図13に示される例と類似である。ステートメント1020は、アダプタ・クラス内のビジネス・メソッドの宣言を示し、整数"i"及び"Employee ee"を引き数として受け取り、Customerを戻り値として返却する。ステートメント1022は、引き数"ee"がタイプ"Employee"として書き置かれることを示す。ステートメント1024は、EmployeeAdapterからCORBAプロキシを獲得するために、"eeAdapter"がラップ解除されることを示す。ステートメント1026は、ビジネス・メソッドが"eeAdapter"を引き数として、プロキシ・オブジェクト上で呼び出されることを示す。なぜなら、CORBAプロキシ・オブジェクトだけがORBを介して受け渡されるべきであるからである。ステートメント1028は、適切なアダプタにより、返却されたプロキシ"custProxy"をラップし、"custAdapter"を獲得することを示す。ステートメント1030は、"custAdapter"がビ

23

ネス・メソッドの呼び出しからの戻り値として、呼び出し側クライアント・オブジェクトに返却されることを示す。このように、アダプタ・コードは、メソッド呼び出しをCORBAプロキシに委託することに加え、必要に応じてデータ変換を実行する。この場合、アダプタ・コードは、ビジネス・メソッド内で引き数として渡されるEJBをラップ及びラップ解除する。更に、アダプタは、ORBインプリメンテーションによりサポートされないデータ・タイプも実行する。

【0076】図17を参照すると、本発明の好適な方法に従い、適切なアダプタによりEJB引き数をラップ及びラップ解除するプロセスのフロー図が示される。プロセスは、クライアント・オブジェクトがビジネス・メソッドを呼び出し、様々な引き数をビジネス・メソッドに受け渡すとき開始する（ステップ1102）。この場合、クライアントは、CORBAサーバ上のリモートEJBのビジネス・メソッドのためのインタフェースを実装する、アダプタ・クラス内のメソッドを呼び出す。アダプタはビジネス・メソッドのために引き数のリストを解析し（ステップ1104）、必要に応じて引き数に対してデータ変換を実行する（ステップ1106）。引き数がEJBを表すか否か、この場合、アダプタによりラップされたEJBのためのCORBAプロキシを表すか否かが判断される（ステップ1108）。肯定の場合、アダプタ・コードがアダプタをEJB引き数からラップ解除し、CORBAプロキシを獲得する（ステップ1110）。

【0077】引き数がEJBでない場合、プロセスはこのメソッド呼び出しにおいて、処理されるべき追加の引き数が存在するか否かを判断する（ステップ1112）。存在する場合、プロセスはステップ1104に戻り、次の引き数を獲得する。もはや引き数が存在しない場合、アダプタはメソッド呼び出しをEJBを渡すCORBAプロキシにORBを介して委託する（ステップ1114）。

【0078】呼び出されたCORBAプロキシに対して、戻り値が存在するか否かが判断される（ステップ1116）。存在しない場合、アダプタは実行を完了し、制御フローをクライアント・コードに返却する（ステップ1124）。戻り値が存在する場合、戻り値がEJBプロキシの場合、アダプタは結果をクライアントに返却する前に、返却されたプロキシを適切なアダプタによりラップする（ステップ1120）。プロセスは次に完了し、クライアント・コードがその実行を継続する（ステップ1124）。

【0079】本発明の利点は、図面に関する前述の詳細な説明から明かであろう。Javaは、Javaクライ

24

アントがリモート・メソッド呼び出し（RMI）と呼ばれる方法を使用して、別のプロセスで実行されるJavaサーバ上のメソッドを呼び出す準備を提供する。しかしながら、CORBA仲間のサーバなど、サーバがJava環境で実行されていない場合、Javaクライアントは、サーバ上のメソッドに対するメソッド呼び出しを発行できない。なぜなら、JavaはCORBAオブジェクトと通信する固有の機構を提供しないからである。JavaクライアントとCORBAサーバ間で、データ整備、すなわち異なるタイプのデータの間のデータ変換を実行する機能的な機構は存在しない。これらの方法は、他のサーバ上で実行される他のEJBに参照を受け渡すステップを含む。Javaクライアントが、CORBAサーバなどの非Java環境で実行されるEJBなどの別のJavaアプリケーションと通信するための標準的な機構は存在しない。

【0080】本発明は、CORBAサーバ内で実行されるEnterprise JavaBean（EJB）のインタフェースを内蔵することにより生成されるアダプタを使用する。これらのアダプタはJavaクライアント側に存在し、それらの内部にEJBを実行するCORBAサーバのリモート・プロキシを保持する。アダプタは、ビジネス・メソッドを呼び出すためにEJBにより指定されるインタフェースを実装するJavaクラスである。アダプタはクライアントからの全てのビジネス・メソッド呼び出しをサーバ上のCORBAプロキシに委託し、JavaクライアントからCORBAプロキシへの、及びその逆のデータ整備を実行する。

【0081】クライアント・オブジェクトの見地から、アダプタは実質的にCORBAサーバ上に存在するEJBのレプリカである。クライアント・オブジェクトは、自身がオブジェクト・リクエスト・ブローカ（ORB）を介してCORBAサーバ上のEJBと通信することを意識しない。アダプタはそれ自身内にEJBを実行するサーバのCORBAプロキシを記憶する。クライアントによりアダプタに発せられるあらゆるビジネス・メソッド呼び出しが適切なデータ変換の依り、アダプタによりCORBAプロキシに委託される。従って、アダプタは、JavaクライアントとCORBAサーバ上のEJBとの間の透過的な接続剤として作用する。

【0082】重要な点は、本発明は完全虚像型のデータ処理システムの状況において述べられてきたが、当業者であれば、本発明のプロセスが命令のコンピュータ読取り可能媒体の形態、及び様々な形態で配布され得、本発明が配布を実施するために実際に使用される特定のタイプの信号担持媒体に関係無しに、同様に当てはまることである。コンピュータ読取り可能媒体の例には、フロッピー・ディスク、ハード・ディスク・ドライブ、RAM、及びCD-ROMなどの記憶型媒体と、デジタル及びアナログ通信リンクなどの伝送型媒体とが含

まれる。

【0083】まとめとして、本発明の構成に関して以下の事項を開示する。

【0084】(1) 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すプロセスであって、前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行するステップと、リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得するステップと、前記オブジェクト参照をアダプタ内でラップするステップと、前記アダプタのメソッドを呼び出すステップとを含む、プロセス。

(2) 前記アダプタが前記オブジェクト参照を使用し、前記サーバ上のスケルトンのメソッドを呼び出す、前記(1)記載のプロセス。

(3) 前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、前記(1)記載のプロセス。

(4) 分散データ処理システムにおいて分散アプリケーションを実装する方法であって、サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得するステップと、前記プロキシをアダプタ内でラップするステップと、前記アダプタのメソッドを呼び出すステップとを含む、方法。

(5) 前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するJavaクラスである、前記(4)記載の方法。

(6) 前記サーバ・オブジェクトがEnterprise JavaBeanである、前記(4)記載の方法。

(7) 前記オブジェクト参照が命名サービスから獲得される、前記(4)記載の方法。

(8) 前記プロキシがCORBAプロキシである、前記(4)記載の方法。

(9) 前記アダプタが前記CORBAプロキシのメソッドを呼び出す、前記(8)記載の方法。

(10) 前記CORBAプロキシがクライアント・コンピュータ上に存在するJavaクラスである、前記(8)記載の方法。

(11) 前記CORBAプロキシがメソッド要求をオブジェクト・リクエスト・ブローカに受け渡す、前記(8)記載の方法。

(12) 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すデータ処理システムであって、前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する実行手段と、リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する獲得手段と、前記オブジェクト参照をアダプタ内でラップするラッピング手段と、前記アダプタのメソッドを呼び出す呼び出し手段とを含む、データ処理システム。

(13) 前記アダプタが前記オブジェクト参照を使用し、前記サーバ上のスケルトンのメソッドを呼び出す、前記(12)記載のデータ処理システム。

(14) 前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、前記(12)記載のデータ処理システム。

(15) 分散データ処理システムにおいて分散アプリケーションを実装するデータ処理システムであって、サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得する獲得手段と、前記プロキシをアダプタ内でラップするラッピング手段と、前記アダプタのメソッドを呼び出す呼び出し手段とを含む、データ処理システム。

(16) 前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するJavaクラスである、前記(15)記載のデータ処理システム。

(17) 前記サーバ・オブジェクトがEnterprise JavaBeanである、前記(15)記載のデータ処理システム。

(18) 前記オブジェクト参照が命名サービスから獲得される、前記(15)記載のデータ処理システム。

(19) 前記プロキシがCORBAプロキシである、前記(15)記載のデータ処理システム。

(20) 前記アダプタが前記CORBAプロキシのメソッドを呼び出す、前記(19)記載のデータ処理システム。

(21) 前記CORBAプロキシがクライアント・コンピュータ上に存在するJavaクラスである、前記(19)記載のデータ処理システム。

(22) 前記CORBAプロキシがメソッド要求をオブジェクト・リクエスト・ブローカに受け渡す、前記(19)記載のデータ処理システム。

(23) 分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すためにデータ処理システム内で使用されるコンピュータ・プログラムを記憶したコンピュータ読取り可能記憶媒体であって、前記コンピュータ・プログラムは、前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する第1の命令と、リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する第2の命令と、前記オブジェクト参照をアダプタ内でラップする第3の命令と、前記アダプタのメソッドを呼び出す第4の命令とを含む、記憶媒体。

(24) 分散データ処理システムにおいて分散アプリケーションを実装するために、データ処理システム内で使用されるコンピュータ・プログラムを記憶したコンピュータ読取り可能記憶媒体であって、前記コンピュータ・プログラムは、サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得する第1の命令と、前記プロキシをアダプタ内でラップする第2の命令と、前記アダプタのメソッドを呼び出す第3の命令と、前記アダプタのメソッドを呼び出す第4の命令とを含む、記憶媒体。

プタのメソッドを呼び出す第3の命令とを含む、記憶媒体。

【図面の簡単な説明】

【図1】本発明が実施される分散データ処理システムを示す図である。

【図2】サーバとして実施されるデータ処理システムを示すブロック図である。

【図3】本発明が実施されるデータ処理システムを示すブロック図である。

【図4】従来の分散アプリケーションを示す図である。

【図5】CORBA規格を使用する従来の分散アプリケーションを示す図である。

【図6】CORBAを用いて相互運用性機能を提供する従来の分散アプリケーション内のコンポーネントを示す図である。

【図7】CORBAサーバ内で実行されるEJBのリモート・ビジネス・メソッドを呼び出す方法を実装するために使用されるコンポーネントを示すブロック図である。

【図8】サーバ・オブジェクトのリモート・メソッドを呼び出す方法を実装するために使用されるコンポーネントを示すブロック図である。

【図9】CORBAサーバ内で実行されるEJB上のリモート・ビジネス・メソッドを呼び出す方法を示すフロー図である。

【図10】JavaクライアントがCORBAサーバ内で実行されるEJBからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Javaプログラミング言語文の例を示す図である。

【図11】JavaクライアントがCORBAサーバ内で実行されるEJBからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Javaプログラミング言語文の例を示す図である。

【図12】JavaクライアントがCORBAサーバ内で実行されるEJBからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Javaプログラミング言語文の例を示す図である。

【図13】JavaクライアントがCORBAサーバ内で実行されるEJBからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Javaプログラミング言語文の例を示す図である。

【図14】Javaメソッドへの引き数として使用されるEJBをラップ及びラップ解除するプロセスを記述するJavaプログラミング言語文の例を示す図である。

【図15】Javaメソッドへの引き数として使用されるEJBをラップ及びラップ解除するプロセスを記述するJavaプログラミング言語文の例を示す図である。

【図16】Javaメソッドへの引き数として使用されるEJBをラップ及びラップ解除するプロセスを記述するJavaプログラミング言語文の例を示す図である。

【図17】本発明の方法に従う好適なアダプタによる、EJB引き数のラップ及びラップ解除のフロー図である。

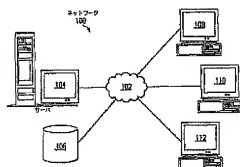
【符号の説明】

100、200、300 分散データ処理システム
102 ネットワーク
104、200、614、726、772 サーバ
106 記憶ユニット
108、110、112、502、700、750 クライアント
202、204、302 プロセッサ
206 システム・バス
208 メモリ制御装置/キャッシュ
209 ローカル・メモリ
210 I/Oバス・ブリッジ
212 I/Oバス
214 周辺コンポーネント相互接続(PCI)バス・ブリッジ
216、306 PCIローカル・バス
218、322 モデム
220 ネットワーク・アダプタ
222、224 PCIバス・ブリッジ
226、228 PCIバス
230 メモリマップド・グラフィックス・アダプタ
232 ハード・ディスク
304 主メモリ
308 PCIブリッジ
310 ローカル・エリア・ネットワーク(LAN)・アダプタ
312 SCSIホスト・バス・アダプタ
314 拡張バス・インタフェース
316 音声アダプタ
318 グラフィックス・アダプタ
319 スマートカード・アダプタ
320 キーボード及びマウス・アダプタ
324 メモリ
326 ハード・ディスク・ドライブ
328 テープ・ドライブ
330 CD-ROMドライブ
400、500、752 クライアント・オブジェクト
402、770 サーバ・オブジェクト
404 データベース
406 Javaリモート・メソッド呼び出し(RMI)プロトコル
408 レガシ・アプリケーション
410 現企業アプリケーション
510 通信リンク
520 CORBAサーバ
521 JavaBean
522 C++オブジェクト

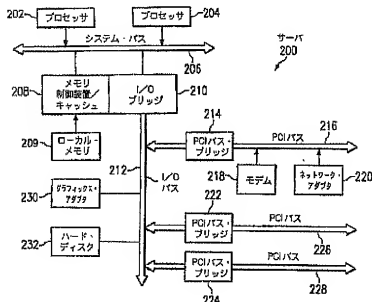
523 Enterprise JavaBean (EJB)
 524 COBOLオブジェクト
 606、610 オブジェクト・リクエスト・ブローカ
 612 スケルトン・コード
 616 オブジェクト参照
 618 メソッド要求
 620 サーバント
 702 Java仮想マシンJVM
 704 Javaクライアント・オブジェクト
 706 アダプタ
 708 EJBインタフェース
 710 CORBAプロキシ
 714、718 オブジェクト・リクエスト・ブローカ

ORB
 716 I I O P
 720 EJBスケルトン
 722 J N I
 724 J V M
 728 リモートEJB
 758 オブジェクト参照
 758 データ整備モジュール
 760、764 オブジェクト・ディスパッチャ
 762 オブジェクト通達リンク
 766 データ整備モジュール
 768 リモート呼び出しスケルトン

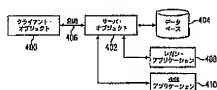
【図1】



【図2】



【図4】



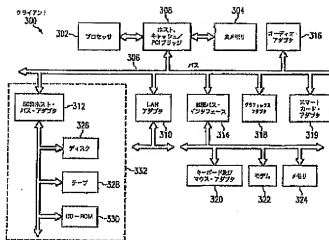
【図13】

```

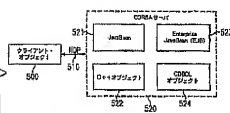
930 ~ public void businessMethod() {
932 ~ proxy.businessMethod(args)
    }

```

【図 3】



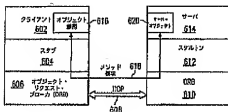
【図 5】



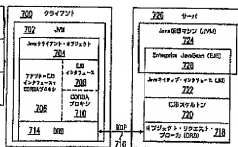
【図 10】

902 → Object obj = java.lang.Class.forName("RemoteObject");
 904 → CustomerInterface cmi = (CustomerInterface) obj;
 906 → cmi.businessMethod(msg);

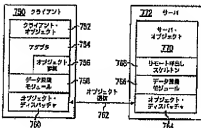
【図 6】



【図 7】



【図 8】



【図 11】

912 → public Object lookup(String serviceName) {
 914 → CustomerProxy proxy = CustomerProxyService.lookup(String serviceName);
 916 → Class adapterClass = Class.forName(proxy.getClassName());
 918 → Adapter obj = new adapterClass(proxy);
 920 → return obj;
 }

【図 12】

922 → Object obj = RMINamingContext.lookup("RemoteObject");
 924 → CustomerInterface cmi = (CustomerInterface) obj;
 926 → cmi.businessMethod(msg);

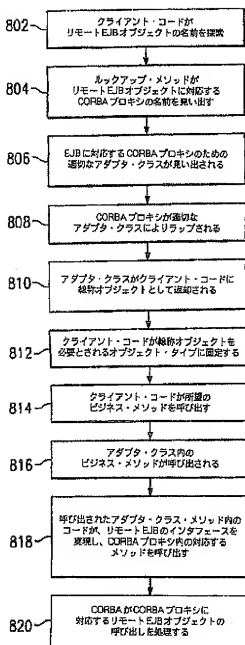
【図 14】

1000 → public Customer businessMethod(Integer i, Employee obj);

【図 15】

1010 → Object obj1 = RMINamingContext.lookup(EmployeeName);
 1012 → Employee emp = (Employee) obj1;
 1014 → Customer cust = (Customer) RMINamingContext.lookup(CustomerName);
 1016 → Customer newCust = cust.businessMethod(i, emp);

【図 9】



【図 16】

```

1020 ~ public Customer unmarshalled (Integer i, Employee ee) {
1022 ~ EmployeeAdapter adapter = (EmployeeAdapter) ac;
1024 ~ EmployeeProxy eProxy = adapter.unmarshal(i);
1026 ~ CustomerProxy cProxy = proxy.unmarshalled(i, eProxy);
1028 ~ CustomerAdapter custAdapter = new CustomerAdapter(cProxy);
1030 ~ return custAdapter;
}
  
```

【図17】

